Database Systems, 236363, winter 2015-2016
Homework#2 - Wet

Online progress report submission until December 3 at 13:30, pairs only.

Online final submission until December 21 at 13:30, pairs only.

Teaching assistant in charge: Roni Licher.

For questions please contact ronili@cs.technion.ac.il with the subject HW2.

In this assignment you are going to implement a program that will work it photos database.

System overview:

The database holds the information of stored photos and tags that describe that photos.

The database tables:

* Underscored names describes the table key

Users(id, name):

- User_id – Integer.
- Name – String.

Photos(id, user_id)

- Id – integer.
- User_id – Integer.

Tags(Photo_id, user_id, info)

- Photo_id – integer.
- User_id – Integer.
- Info – String.

Interface

- You are required to implement the following interface that mange the db.
- You may assume that the input is correct.
- You must not assume anything about the tables contact in anytime. You can only assume that if a table contains data, it obeys the table constrains and requirements.
- You are required to check the error cases described in the assignments.
- You may assume that there are not connection errors between the client and the db. Yet, you should check each command result status and in case of failure, print an informative message.
- In case that a function that consist of multiple SQL commands fails because it derives from the assignment requirement, you should make sure you don't leave the DB in a partial mutation state.
- The program input is the STDIN. The commands parameters are separated by space.
- All the constant strings are defined in the wet.h. All your prints should use only the defined strings and the tables' data.
- All output should be made to the STDOUT, no exceptions.
- The next section describes each function that you have to implement:

1. Adding a user:
   Input:
   addUser name
   Description:
   The command adds a user and provides a new user id. The user id will be bigger in 1 than the maximal user id available in the table.
   Output and Comments:
   - The program should output the new id using the constant ADD_USER.
   - The minimal number should be 0 (zero). No special additional queries should be made to the DB in order to check this status. You should integrate this check in the main command.
   - You should implement this function using 1 SQL command for the insert. After the insert you should query the DB and get the new ID.

2. Adding a user in the minimal position available.
   Input:
   addUserMin name
   Description:
   This command lets you insert users with ids of removed users from the system. The command adds a user and provides a new user id. The user id will be the minimal id available in the system that is greater than an existing available user id (this strange requirement was made in order to make it easier to implement).

Example: Let's say we have a table with the following user keys: 1,3,5,7. The new id will be 2.

Output and Comments:

- You should print the table header USER_HEADER and then all of the users that has the input name with the USER_RESULT constant.
- The contact of the table will be sorted by the user id ascending.
- If no users in the table, the minimal number should be 0 (zero). No special additional queries should be made to the DB in order to check this status. You should integrate this check in the main command.

3. Removing a user:

Input:

removeUser id

Description:

The command removes a user (if exists), all his photos and all of the photos tags.

Output and Comments:

- In case that the user doesn't exist, you should print the constant ILL_PARAMS.
- In case of success, you should not print anything.

4. Adding a photo:

Input:

addPhoto user_id, photo_id

Description:

The command adds a photo for a user.

Output and Comments:

- In case that the user doesn't exist, you should print the constant ILL_PARAMS.
- In case that the user already has a photo with this id, you should print the constant EXISTING_RECORD.
- In case of success, you should not print anything.

5. Adding a tag:

Input:

tagPhoto user_id, photo_id, info

Description:

The command tags the user's photo by adding some info.

Output and Comments:

- In case that the user don't exist, you should print the constant ILL_PARAMS.
- Assume that the photo exists or that the trigger from command 11 is installed.
- You should place the tag even if the photo doesn't exist.
- In case that the exact same tag already exists for the user and photo, you should print the constant EXISTING_RECORD.

- In case of success, you should not print anything.

6. Photos tags
   Input:
   photosTags
   Description:
   The command returns the number of tags for each photo.
   Output and Comments:
   - For each photo, the number of tags will be printed.
   - You should print the table header PHOTOS_HEADER and then all of the photos and counters with the PHOTOS_RESULT constant.
   - In case of empty results, the EMPTY constant will be printed.
   - The contact of the table will be sorted by:
     - First: The number of tags, descending.
     - Second: The user_id, ascending.
     - Third: The photo id, ascending.
   - In this command you are not allowed to use subqueries and\or set operators (such as union).

7. Searching photos:
   Input:
   search word
   Description:
   The command returns all of the photos that has a tag that contains this word.
   Output and Comments:
   - The header PHOTOS_HEADER and then all of the photos that has a tag with this word will be printed with the PHOTOS_RESULT constant.
   - For each photo, the number of tags will be printed.
   - In case of empty results, the EMPTY constant will be printed.
   - The contact of the table will be sorted by:
     - First: The number of tags, descending.
     - Second: The user_id, ascending.
     - Third, the photo_id, descending.

8. Common tags:
   Input:
   commonTags k
   Description:
   The command returns tags that appear identically for at least k photos.
   Output and Comments:

- The header COMMON_HEADER and then the tags will be printed with the COMMON_RESULT constant.
- For each tag, the number of photos will be printed.
- In case of empty results, the EMPTY constant will be printed.
- The contact of the table will be sorted by:
  - First: The number of photos, descending.
  - Second: The tag, lexicographically ascending.
- Assume $k \geq 1$.

9. Common tags:
   Input:
   mostCommonTags k
   Description:
   The command returns the k tags that appear the most.
   * Using the Limit operator is forbidden here.
   Output and Comments:
   - Tie breaker for tags that appear the same number of times will be the lexicographically order, the smaller will have higher score.
   - The header COMMON_HEADER and then the tags will be printed with the COMMON_RESULT constant.
   - For each tag, the number of photos will be printed.
   - In case of empty results, the EMPTY constant will be printed.
   - The contact of the table will be sorted by:
     - First: The number of tags, descending.
     - Second: The tag, lexicographically ascending.
   - Assume $k \geq 1$.

10. Similar photos:
    Input:
    similarPhotos k j
    Description:
    The command returns the photos that has at least **k** other photos that each has at least **j** identical tags as the photo.
    Output and Comments:
    - The header SIMILAR_HEADER and then the tags will be printed with the SIMILAR_RESULT constant.
    - In case of empty results, the EMPTY constant will be printed.
    - The contact of the table will be sorted by:
      - First: User id, ascending.
      - Second: Photo id, ascending.
    - Assume $k, j \geq 1$.

11. Add trigger:
    Input:
        autoPhotoOnTagOn
    Description:
        The command creates a trigger on the 'Tag' table. This trigger will be used to add a photo for a user (only if he exists) when a tag is added to a photo that is not in the db.
    Output and Comments:
    - You should not print anything.
    - In order to be able to work with triggers you should run the next SQL command once: createlang -d dbname plpgsql. Do not put it in you code, use the web interface to run it.

12. Remove trigger:
    Input:
        autoPhotoOnTagOff
    Description:
        The command removes that trigger from the add trigger command.
    Output and Comments:
    - You should not print anything.

Main

You are required to create the main function. Use it to create the DB connection handler and to call the provided *parseInput* function.

More requirements:

1. You should implement the program in ANSI C.
2. It must compile without errors on the CSL2 server.
3. The program will use the PostgreSQL LIBPQ library (installed on the CSL2).
4. You should implement the solution in one file, wet.c.

Supplied files:

1. Wet.h – Contains strings and functions declarations.
2. Parser.c – Contains the parseInput function. Use it to parse the commands.
3. Example.sol & Example.in – Basic tests.
4. Makefile.

Submit

- One C file, wet.c. It should be in unix format. Make sure it compiles on the CSL2 before submitting. You will might need to use dos2unix.

Connecting to the DB:

- The only lines you have to change in the wet.h file is the next 2:

  #define **USERNAME** "user"

  #define **PASSWORD** "pass"

  Use your CSL2 username (t2, tx user name) and the password is your id. (You must create an account first according to the instruction sent earlier this semester)

  DO NOT CHANGE ANYTHING ELSE IN THAT FILE.

- In the wet.c file add the #include "wet.h" line.
- In the main function use the next lines in order to connect:

```
char connect_param[200];

sprintf(connect_param,

        "host=csl2.cs.technion.ac.il dbname=%s user=%s password=%s",

        USERNAME, USERNAME, PASSWORD);

PGconn *conn = PQconnectdb(connect_param);
```

- You can use the examples in: http://webcourse.cs.technion.ac.il/236363/Winter2015-2016/ho/WCFiles/PostgreSQL_s2014.pdf
- Compile the program with the makefile or with the next command:
  gcc -I/usr/local/pgsql/include -L/usr/local/pgsql/lib -lpq -o wet wet.c parser.c

Tables

The tables will be soon updated in your account. Those tables can be read and modified, it is your private copy. In addition you will have tables with the “course_” prefix. Those are backup tables with read only permissions.

In order to recover the original data, you can run the next commands.

```
DROP TABLE tableName;
CREATE TABLE tableName AS SELECT * FROM course_tableName;
```

On every run the automatic test will reset the table’s data.

DO NOT put those lines in your code.

Testing your program with the basic test

Just run the next commands:

./wet < Example.in >& Example.out
diff –w Example.out Example.sol

- Remember, we assume the tables are in their initial state.

Using SQL and C code:

You should keep in mind that this is an SQL assignment and not a C assignment. We expect you to do everything possible in SQL and only to wrap it with C code in order to be able to run those.

That's mean:

- You cannot process the queries output. You should print the results as is!
- You cannot omit some of the results with C code. You should print all of the lines returned.
- You should run as few SQL commands as possible.
- Querying the DB in a loop is definitely a no go.
- You are not allowed to transfer any data between queries. In example, reading some number from the DB and then putting it in a new query is wrong.

Programing Guidelines

- You should release any resource allocated (connection, result, views …).
- You may use a view only if you use it at least twice.
- You should test your program well.
- You should not stress test your program. This is a shareable server for all of the students and it is being monitored.
- We expect you to write good, high standard code. Poor, obscure code will result in deduction of points.

Grades

- The assignment will be scored based on automatic testing and manual checking.
- Every semester, some students are caught cheating. We are forwarding all involved assignments to the Technion discipline committee.
- Using C code instead of SQL code will reduce points heavily.
- Using unnecessary complicated SQL commands may reduce points.
- No progress submission -> No final score.

Progress Submission Report

You have to submit a progress report by December 3. In the report you have to write the progress you have done in solving the assignment, if any (that's acceptable).

It should be one txt or pdf file that contains:

- The pair id's, names and emails.
- Status of your csl2 account, **must** be in this point one of the next options:
    - Working fine.
    - Has issues but notified the teaching assistant in charge.
- The progress.
- Anything you wish to say or ask.

Submitting this report is **mandatory.**

Good Luck!